Automating health economic evaluation with GitHub Actions & plumber.

Robert Smith[1,2,3,4], Paul Schneider[1,2] & Wael Mohammed[1,2]

1 Dark Peak Analytics, Sheffield
2 ScHARR, University of Sheffield
3 Lumanity, Sheffield
4 UK Health Security Agency

Disclaimer

The views expressed in this presentation are that of the author, and not the affiliated institutions.

Competing interests: R.A.S. Is part of the Scientific Committee for R for HTA, an academic consortium whose main objective is to explore the use of R for cost-effectiveness analysis. P.P.S. and W.M have no competing interests to declare.

Grant information: This work was jointly supported by the Wellcome Trust Doctoral Training Centre in Public Health Economics and Decision Science (PHEDS) [108903, https://doi.org/10.35802/108903; 224853] and the University of Sheffield.
The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

How to cite: Smith RA, Schneider PP and Mohammed W. Living HTA: Automating Health Technology Assessment with R [version 1; peer review: 1 approved with reservations]. Wellcome Open Res 2022, 7:194 (https://doi.org/10.12688/wellcomeopenres.17933.1)

Content

- What is Health Economic Evaluation?
- Current process for health economic models
- Future process for health economic models
- Previous work: Making Health Economics Shiny
- This work: Automating Health Economic Evaluation
    1. *plumber* script used to generate an API.
    2. R script which uses 1, 2 and 3 and *Rmarkdown* to generate a 'living HTA' report.
    3. *GitHub actions workflow* which automates this process monthly.
    4. *R-Shiny app* which allows non-technical users to query the API.

What is Health Economic Evaluation?

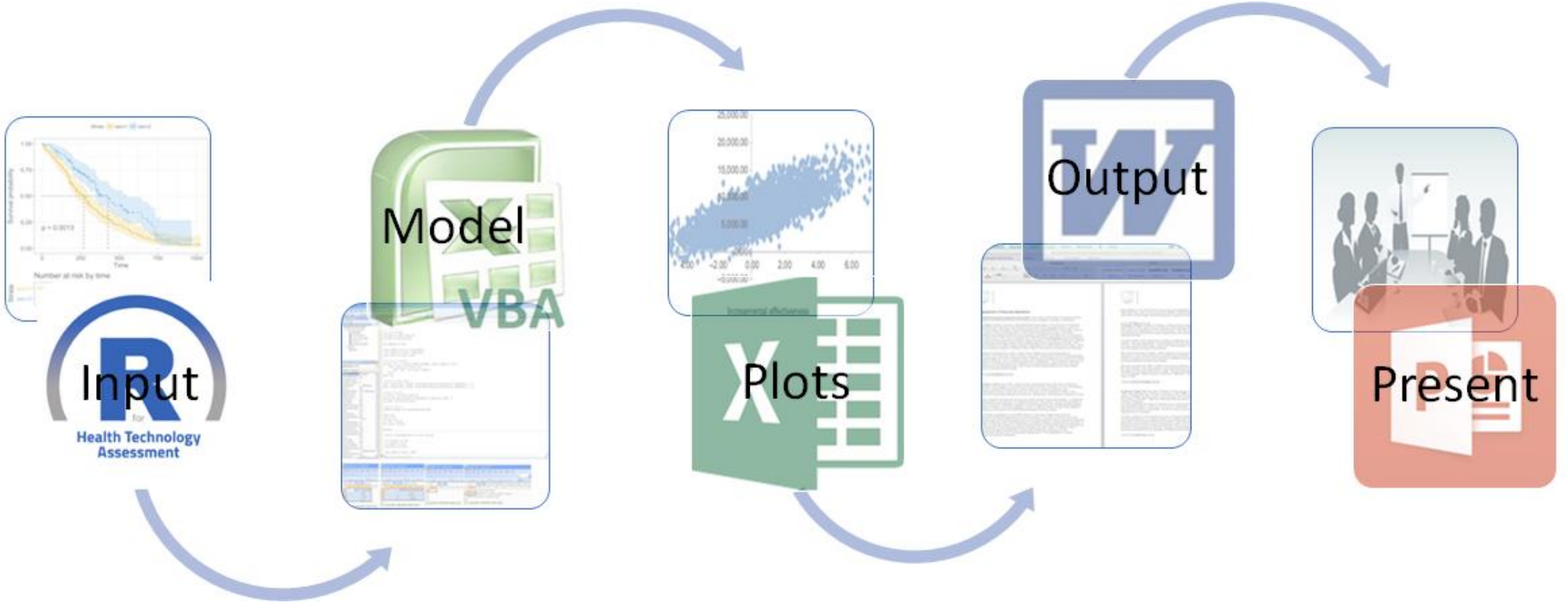
"[HTA is] a multidisciplinary process that aims to determine the value of a health technology and to inform guidance on how these technologies can be used in health systems around the world."

World Health Organisation

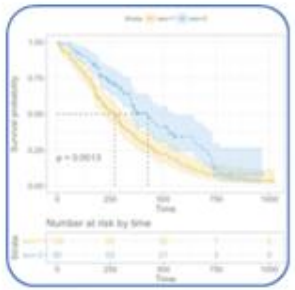
"The term health economic evaluation describes the comparative assessment of costs and outcomes of alternative health care technologies or health strategies"

Hessel, F. (2008)

Current process for health economic models

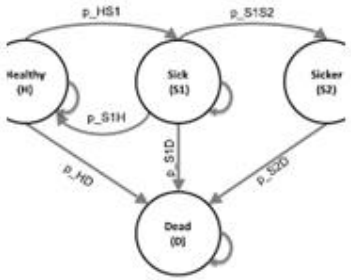


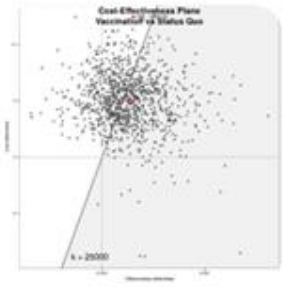
Future process for health economic models



R for Health Technology Assessment

hesim health economic simulation modeling





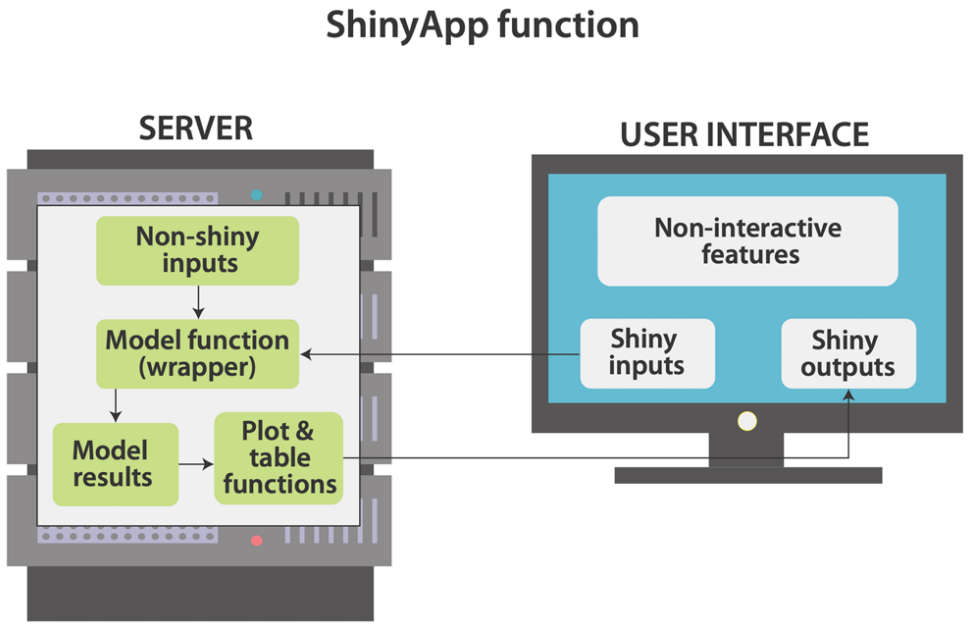
ggplot2

rmarkdown

Shiny

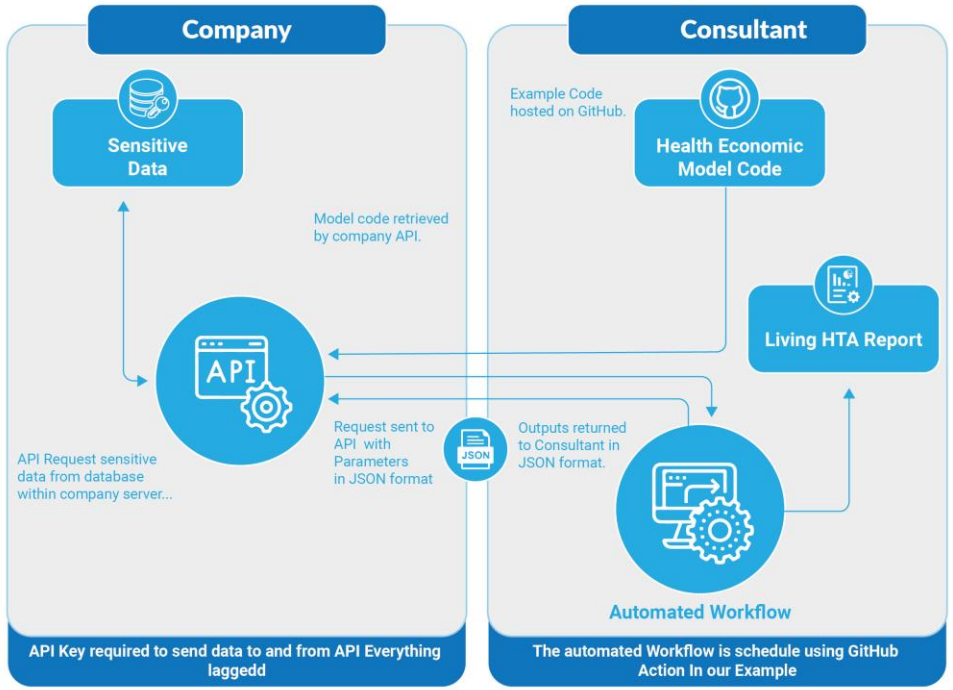
https://r-hta.org/

Making Health Economics Shiny



App: https://robertasmith.shinyapps.io/sick_sicker/
Paper: https://wellcomeopenresearch.org/articles/5-69
Code: https://github.com/RobertASmith/paper_makeHEshiny
Tutorial: https://r-hta.org/tutorial/markov_models_shiny/

Making Health Economics … hAPpI



Wellcome Open Research — Wellcome Open Research 2022, 7:194 Last updated: 08 AUG 2022

Check for updates

METHOD ARTICLE

Living HTA: Automating Health Technology Assessment with R [version 1; peer review: 1 approved with reservations]

Robert A. Smith[1-3], Paul P. Schneider[1,3], Wael Mohammed[1,3]

[1]School of Health and Related Research, University of Sheffield, Sheffield, S1 4DA, UK
[2]Lumanity, Sheffield, S1 2GQ, UK
[3]Dark Peak Analytics, Sheffield, S11 7BA, UK

V1 **First published:** 21 Jul 2022, **7**:194
https://doi.org/10.12688/wellcomeopenres.17933.1
Latest published: 21 Jul 2022, **7**:194
https://doi.org/10.12688/wellcomeopenres.17933.1

Abstract

Background: Requiring access to sensitive data can be a significant obstacle for the development of health models in the Health Economics & Outcomes Research (HEOR) setting. We demonstrate how health economic evaluation can be conducted with minimal transfer of data between parties, while automating reporting as new information becomes available.
Methods: We developed an automated analysis and reporting pipeline for health economic modelling and made the source code openly available on a GitHub repository. The pipeline consists of three parts: An economic model is constructed by the consultant using pseudo data. On the data-owner side, an application programming interface (API) is hosted on a server. This API hosts all sensitive data, so that data does not have to be provided to the consultant. An automated workflow is created, which calls the API, retrieves results, and generates a report.
Results: The application of modern data science tools and practices allows analyses of data without the need for direct access – negating the need to send sensitive data. In addition, the entire workflow can be largely automated: the analysis can be scheduled to run at defined time points (e.g. monthly), or when triggered by an event (e.g. an update to the underlying data or model code); results can be generated automatically and then be exported into a report. Documents no longer need to be revised manually.
Conclusions: This example demonstrates that it is possible, within a HEOR setting, to separate the health economic model from the data, and automate the main steps of the analysis pipeline.

Keywords
HEOR, HTA, APIs, R, plumber

Open Peer Review

Approval Status ?

	1
version 1 21 Jul 2022	? view

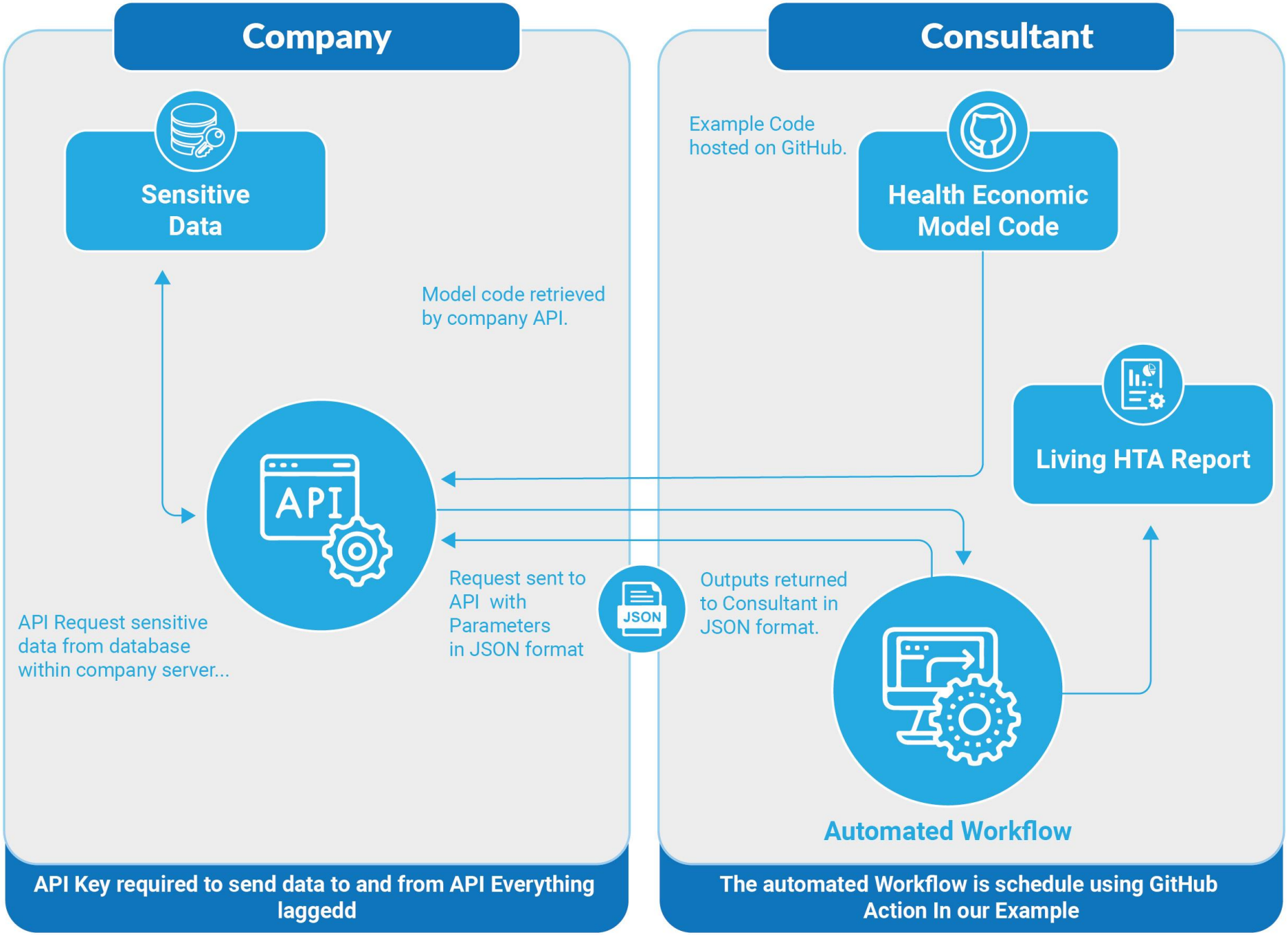
1. **Mohsen Sadatsafavi**, University of British Columbia, Vancouver, Canada

Any reports and responses or comments on the article can be found at the end of the article.

App: https://rasmithbresmed.shinyapps.io/living_hta/
Paper: https://wellcomeopenresearch.org/articles/7-194
Code: https://github.com/RobertASmithBresMed/plumberHE

What I will show today

- GitHub repository containing:
  1. R Script containing a health economic model written in R & C++.
  2. *plumber* script used to generate an API.
  3. *Rmarkdown* document which is used to generate a model report.
  4. R script which uses 1, 2 and 3 to update a report without access to data.
  5. *GitHub actions workflow* which automates this process monthly.
  6. *R-Shiny app* which allows non-technical users to query the API.


Company
Consultant
Sensitive Data
Example Code hosted on GitHub.
Health Economic Model Code
Model code retrieved by company API.
Living HTA Report
API
API Request sensitive data from database within company server...
Request sent to API with Parameters in JSON format
JSON
Outputs returned to Consultant in JSON format.
Automated Workflow
API Key required to send data to and from API Everything laggedd
The automated Workflow is schedule using GitHub Action In our Example

What I will show today

- GitHub repository containing:
  1. R Script containing a health economic model written in R & C++.
  2. ***plumber* script used to generate an API.**
  3. *Rmarkdown* document which is used to generate a model report.
  4. **R script which uses 1, 2 and 3 to update a report without access to data.**
  5. *GitHub actions workflow* which automates this process monthly.
  6. *Shiny app* which allows non-technical users to query the API.

https://github.com/RobertASmithBresMed/plumberHE

Hosting a health economic model using plumber

Code chunk 1 - Generating the API

```
library(dampack)
library(readr)
library(assertthat)

#* @apiTitle Client API hosting sensitive data
#*
#* @apiDescription This API contains sensitive data, the client does not
#* want to share this data but does want a consultant to build a health
#* economic model using it, and wants that consultant to be able to run
#* the model for various inputs
#* (while holding certain inputs fixed and leaving them unknown).

#* Run the DARTH model
#* @serializer csv
#* @param path_to_psa_inputs is the path of the csv file containing the PSA parameters
#* @param model_functions gives the GitHub repository to source the model code
#* @param param_updates gives the replacement values of the editable parameters
#* @post /runDARTHmodel
function(path_to_psa_inputs = "parameter_distributions.csv",
         model_functions = paste0("https://raw.githubusercontent.com/",
                                  "BresMed/plumberHE/main/R/darth_funcs.R"),
         param_updates = data.frame(
           parameter = c("p_HS1", "p_S1H"),
           distribution = c("beta", "beta"),
           v1 = c(25, 50),
           v2 = c(150, 70)
         )) {


  # source the model functions from the shared GitHub repo...
  source(model_functions)

  # read in the csv containing parameter inputs
  psa_inputs <- as.data.frame(readr::read_csv(path_to_psa_inputs))

  # for each row of the data-frame containing the variables to be changed...
  for(n in 1:nrow(param_updates)){

  # update parameters from API input
  psa_inputs <- overwrite_parameter_value(
                              existing_df = psa_inputs,
                              parameter = param_updates[n,"parameter"],
                              distribution = param_updates[n,"distribution"],
                              v1 = param_updates[n,"v1"],
                              v2 = param_updates[n,"v2"])
  }

  # run the model using the single run-model function.
  results <- run_model(psa_inputs)

  # check that the model results being returned are the correct dimensions
  # here we expect a single dataframe with 6 columns and 1000 rows
  assertthat::assert_that(
    all(dim(x = results) == c(1000, 6)),
    class(results) == "data.frame",
    msg = "Dimensions or type of data are incorrect,
  please check the model code is correct or contact an administrator.
  This has been logged"
  )

  # check that no data matching the sensitive csv data is included in the output
  # searches through the results data-frame for any of the parameter names,
  # if any exist they will flag a TRUE, therefore we assert that all = F
  assertthat::assert_that(all(psa_inputs[, 1] %in%
       as.character(unlist(x = results,
                           recursive = T)) == F))

  return(results)

}
```

Load necessary packages

Describe the API, used in Swagger

Roxygen style documentation for function, what are the inputs…

Source the model functions from GitHub

Overwrite default data with non-sensitive inputs

Run the model

Check the results object doesn't contain sensitive data

Return the results object

Schloerke B, Allen J (2022). *plumber: An API Generator for R*. https://www.rplumber.io.

Hosting a health economic model using plumber (1)

```
#* @apiTitle Client API hosting sensitive data
#*
#* @apiDescription This API contains sensitive data, the client does not
#* want to share this data but does want a consultant to build a health
#* economic model using it, and wants that consultant to be able to run
#* the model for various inputs
#* (while holding certain inputs fixed and leaving them unknown).

#* Run the DARTH model
#* @serializer csv
#* @param path_to_psa_inputs is the path of the csv file containing the PSA parameters
#* @param model_functions gives the GitHub repository to source the model code
#* @param param_updates gives the replacement values of the editable parameters
#* @post /runDARTHmodel
function(path_to_psa_inputs = "parameter_distributions.csv",
         model_functions = paste0("https://raw.githubusercontent.com/",
                                  "BresMed/plumberHE/main/R/darth_funcs.R"),
         param_updates = data.frame(
           parameter = c("p_HS1", "p_S1H"),
           distribution = c("beta", "beta"),
           v1 = c(25, 50),
           v2 = c(150, 70)
         )) {
```

Describe the API, used in Swagger

Roxygen style documentation for function, what are the inputs…

Schloerke B, Allen J (2022). *plumber: An API Generator for R*. https://www.rplumber.io.

Hosting a health economic model using plumber (2)

```
30    # source the model functions from the shared GitHub repo...
31    source(model_functions)
32
33    # read in the csv containing parameter inputs
34    psa_inputs <- as.data.frame(readr::read_csv(path_to_psa_inputs))
35
36    # for each row of the data-frame containing the variables to be changed...
37    for(n in 1:nrow(param_updates)){
38
39    # update parameters from API input
40    psa_inputs <- overwrite_parameter_value(
41                              existing_df = psa_inputs,
42                              parameter = param_updates[n,"parameter"],
43                              distribution = param_updates[n,"distribution"],
44                              v1 = param_updates[n,"v1"],
45                              v2 = param_updates[n,"v2"])
46    }
47
```

Source the model functions from GitHub

Overwrite default data with non-sensitive inputs

Schloerke B, Allen J (2022). *plumber: An API Generator for R*. https://www.rplumber.io.

Hosting a health economic model using plumber (3)

```
48    # run the model using the single run-model function.
49    results <- run_model(psa_inputs)
50
51    # check that the model results being returned are the correct dimensions
52    # here we expect a single dataframe with 6 columns and 1000 rows
53    assertthat::assert_that(
54      all(dim(x = results) == c(1000, 6)),
55      class(results) == "data.frame",
56      msg = "Dimensions or type of data are incorrect,
57    please check the model code is correct or contact an administrator.
58    This has been logged"
59    )
60
61    # check that no data matching the sensitive csv data is included in the output
62    # searches through the results data-frame for any of the parameter names,
63    # if any exist they will flag a TRUE, therefore we assert that all = F
64    assertthat::assert_that(all(psa_inputs[, 1] %in%
65          as.character(unlist(x = results,
66                              recursive = T)) == F))
67
68    return(results)
69
70  }
```

Run the model

Check the results object doesn't contain sensitive data

Return the results object

Schloerke B, Allen J (2022). *plumber: An API Generator for R*. https://www.rplumber.io.

Running the model – calling the API

Code chunk 2 - Query the API, retrieve model results and generate report

```r
# remove all existing data from the environment.
rm(list = ls())

library(ggplot2)
library(jsonlite)
library(httr)

# run the model using the connect server API
results <- httr::content(
  httr::POST(
    # the Server URL can also be kept confidential, but will leave here for now
    url = "https://connect.bresmed.com",
    # path for the API within the server URL
    path = "rhta2022/runDARTHmodel",
    # code is passed to the client API from GitHub.
    query = list(model_functions =
                   paste0("https://raw.githubusercontent.com/",
                          "BresMed/plumberHE/main/R/darth_funcs.R")),
    # set of parameters to be changed ...
    # we are allowed to change these but not some others
    body = list(
      param_updates = jsonlite::toJSON(
        data.frame(parameter = c("p_HS1","p_S1H"),
                   distribution = c("beta","beta"),
                   v1 = c(25, 50),
                   v2 = c(150, 100))
      )
    ),
    # we include a key here to access the API here the key is a env variable
    config = httr::add_headers(Authorization = paste0("Key ",
                                                      Sys.getenv("CONNECT_KEY")))
  )
)

# write the results as a csv to the outputs folder...
write.csv(x = results,
          file = "outputs/darth_model_results.csv")

source("report/makeCEAC.R")
source("report/makeCEPlane.R")

# render the markdown document from the report folder,
# passing the results dataframe to the report.
rmarkdown::render(input = "report/darthreport.Rmd",
                  params = list("df_results" = results),
                  output_dir = "outputs")
```

Load necessary packages

Call the API:

- URL and path combine to identify where the API is.
- Query and body both allow for inputs to be provided to the API… We convert the data-frame of inputs to JSON first.
- Config allows us to add the KEY – which is hidden as an environment variable.
- **Result of the API is stored as an object** (results).

Write the results to a csv… not strictly necessary.

Render an Rmarkdown document based on the results of the API call, store the document in ‘outputs’ directory.

Automating health economic model updates with GitHub Actions

Code chunk 3 - Automated report updates

```yaml
on:
  push:
    branches:
    - main
  schedule:
    - cron: '1 1 1 * *'

name: Run DARTH model on client API
jobs:
  createPullRequest:
    runs-on: windows-2019
    env:
      GITHUB_PAT: ${{ secrets.GITHUB_TOKEN }}
  # Load repo and install R
    steps:
    - uses: actions/checkout@master
    - uses: r-lib/actions/setup-r@master

    - name: Setup pandoc
      uses: r-lib/actions/setup-pandoc@v2
      with:
        pandoc-version: '2.17.1.1'

    - name: Install TinyTeX
      uses: r-lib/actions/setup-tinytex@v2
      env:
          # install full prebuilt version
          TINYTEX_INSTALLER: TinyTeX

    - name: Install dependencies
      run: |
          install.packages(
          c("reshape2", "jsonlite", "httr", "readr", "rmarkdown", "markdown")
          )
          install.packages(
          "scales", dependencies = TRUE, repos = 'http://cran.rstudio.com/'
          )
          install.packages(
          "ggplot2", dependencies = TRUE, repos = 'http://cran.rstudio.com/'
          )
      shell: Rscript {0}

    - name: Run the model from API and create report
      env:
         CONNECT_KEY: ${{secrets.PLUMBER_SECRET}}
      run: |
          source("scripts/run_darthAPI.R")
      shell: Rscript {0}

    - name: Create Pull Request
      uses: peter-evans/create-pull-request@v3
      with:
        token: ${{ secrets.GITHUB_TOKEN }}
        commit-message: Automated Model Run from API
        title: 'Living HTA Automated Model Run'
        body: >
          Automated model run
        labels: report, automated pr
```

Schedule jobs based upon a **push to the main branch, or at a scheduled time** (00:01 on 1st of the month).

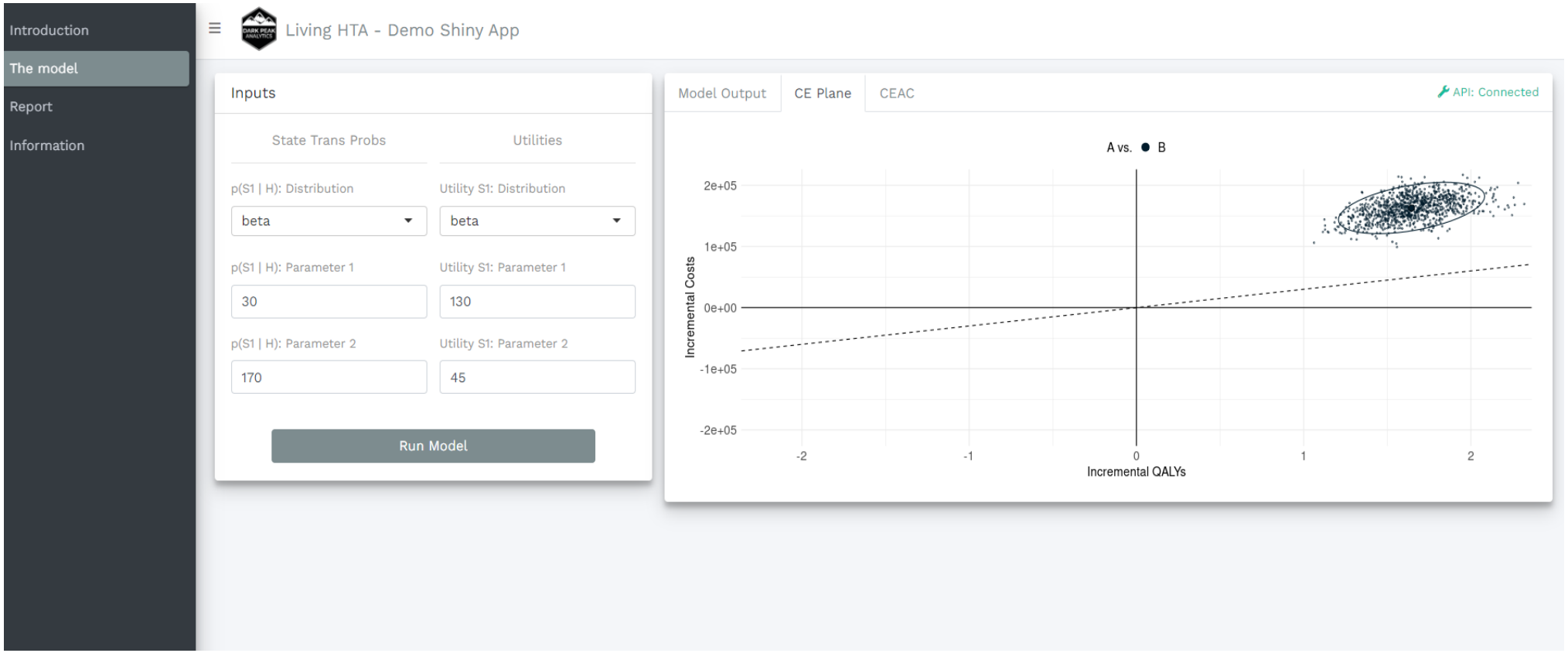
Set-up code:

- Start running a Windows 2019 server.
- Checkout the repository.
- Set up R.
- Install all dependencies

Run the script querying the API and creating a report with Rmarkdown.

Create a pull request to the repository with the new results csv and markdown report included.

R-Shiny app



https://rasmithbresmed.shinyapps.io/living_hta/

What are the pros and cons of this framework.

Advantages of this framework

- **Security -** Data owners retain control of their data. No data need leave the data-owner's servers.
- **Transparency -** Separating the model code from the data can significantly improve the transparency of the health economic model. Many models could be passed to the data, not just one!
- **Computational Power** - The computational burden of the model is handled on a remote server.
- **Storage** – Larger datasets can be analyzed than would be possible on a laptop.
- **Living analysis** - API calls can be made at any time. A decision maker can see a report that will always reflect the data held by the company.

Disadvantages of this framework

- **Security -** Likely to remain concerns about data security, even with the authentication procedures built into the API functionality.
- **Transparency -** Risk that running the model remotely will result in the perception that the model is a 'black box' (I'd disagree!).
- **Coding practice -** The model code needs to be versatile enough to manage unknown data updates. *Proper testing will help mitigate these risks.*
- **Technical skillset -** This is not commonly implemented, or a common skill-set among health economists. Most models are not built in R.

Who can access what?

Agent	Sensitive Data	Model code	Other data
Data Owner (Pharmaceutical company)	☑	☑	☑
External Consultant (Health Economist)	x	☑	☑
3rd Party Consultant (App designer)	x	x	☑

Further resources

More information about this presentation can be found at:

Open-source code: RobertASmithBresMed/plumberHE: Health Economics using Plumber APIs (github.com)
Open access paper: https://wellcomeopenresearch.org/articles/7-194
Open-access app: https://rasmithbresmed.shinyapps.io/living_hta/
R package *plumber*: https://www.rplumber.io
Health economic model code adapted from: https://github.com/DARTH-git

More information about the authors' organizations can be found at:
Dark Peak Analytics
ScHARR, University of Sheffield
Lumanity

The views expressed in this presentation are that of the author and not the institutions…